DOI: 10.15439/2015F322
ACSIS, Vol. 5

Measuring the performance and energy consumption of AES in wireless sensor networks

Cristina Panait
Faculty of Automatic Control and Computers
University POLITEHNICA of Bucharest
Email: cristina.panait19@gmail.com

Dan Dragomir
Faculty of Automatic Control and Computers
University POLITEHNICA of Bucharest
Email: dan.dragomir@cs.pub.ro

***Abstract*—With WSN deployments increasing in popularity, securing those deployments becomes a necessity. This can be achieved by encrypting inter-node communications and/or using message authentication codes. AES is a well studied symmetric cipher, with no known practical vulnerabilities, that can be used to solve both problems. We provide an optimized implementation of AES, with four modes of operation (ECB, CBC, CFB and CTR), that uses the hardware accelerator available on the ATmega128RFA1 microcontroller, and compare it with the best known software implementation. We show that our hardware AES implementation is both faster and more energy efficient than a software implementation. This is not the case for previous sensor nodes and implementations, which show an improved execution speed, but with a higher energy consumption.**

I. Introduction

AS A general definition, a wireless sensor network is composed of a set of nodes which communicate through a wireless medium in order to perform certain tasks. A couple of examples where WSNs can be deployed, as stated in [1], are: fire extension detection, earthquake detection, environment surveillance for pollution tracking, intelligent building management, access restriction, detection of free spaces in parking lots and so on. Advantages brought by WSNs are enhanced flexibility and mobility, mainly because nodes are generally powered from an on-board battery and thus do not depend on their surroundings. This, however, is also their biggest weakness. The lifetime expectancy of a node depends on its usage. The constraints mainly come from the limited energy source, as data processing and transmission can be energy intensive.

The particular characteristics of these types of networks make the direct implementation of conventional security mechanisms difficult. The imposed limitations on minimizing data processing and storage space and reducing bandwidth need to be addressed. The major constraints for WSNs, as presented in [2], [3] and [4], are: energy consumption (which can lead to premature exhaustion of the energy source and to the denial of service), memory limitations (flash, where the application source code is stored, and RAM, where sensed data and intermediary computing results are stored), unreliable communication (the routing protocols used, collisions), latency (which can lead to synchronization issues and algorithms that cannot act correctly) and unattended nodes (an attacker could have physical access to the nodes).

The concept behind WSNs and their applications presents an increased risk to a series of attacks which can affect the network's functionality. In this paper we analyze algorithms that provide confidentiality for WSNs. We focus our analysis on AES-128, as it is a well studied cipher with no known practical vulnerabilities, has a speed comparable with other symmetrical encryption algorithms and is supported on multiple WSN platforms through a hardware acceleration module.

In section II we discuss some of the related work. Sections III and IV present the algorithm design and modes of operation and the implementation with two methods, software and hardware. Then, in section V, we make a comparative analysis of the solutions, based on execution time and energy consumption, and select the encryption methods suitable for ATmega128RFA1-based platforms, taking also into consideration the provided security. Finally, we present the conclusions of our work.

II. Related Work

The problem of measuring the cost of encryption on wireless sensor node hardware has been addressed previously. In [5] Lee et al. analyze a range of symmetric-key algorithms and message authentication algorithms in the context of WSNs. They use the MicaZ and TelosB sensor nodes and measure the execution time and energy consumption of different algorithms. For AES they provide measurements for a hardware assisted implementation and conclude that it is the cheapest when either time or energy is considered. They do not however study this implementation on different plaintext lengths and instead rely on datasheets to extend to lengths longer than one block. However, this conclusion is not backed by Zhang et al. in [6] which compares different AES implementations on the MicaZ nodes. They conclude that hardware assisted encryption is faster, but also consumes more energy due to the external chip which handles the computation in hardware.

Compared to their work, we study only AES-128 which is a well known cipher also adopted by the National Institute of Standards and Technology (NIST) and which has been proposed as a viable alternative ([7]) to other less studied ciphers in WSN applications. This choice is also supported by the fact that multiple 802.15.4 transceivers offer a hardware accelerator for AES operations. We study the newer Sparrow v3.2 sensor nodes based on the ATmega128RFA1, which integrates the microcontroller with the radio transceiver and hardware encryption module, and show that AES-128 can be efficiently implemented reducing both execution time and energy consumption. We also provide hybrid implementations for modes of operation that are not natively supported by the hardware and show that they can still be efficiently implemented with the available primitives.

In [7] Law et al. conduct a thorough survey of the costs of different block ciphers, when implemented on sensor node hardware. They conclude that Rijndael (AES) is the second most efficient cipher, being surpassed only by Skipjack. However, their analysis is based on older hardware and does not consider any hardware accelerated implementations.

In [8] de Meulenaer et al. study the problem of key exchange and measure the cost of two key agreement protocols: Kerberos and Elliptic Curve Diffie-Hellman. They measure the energy consumption of the two protocols on MicaZ and TelosB sensor nodes and conclude that the listening mode is the principal factor in the energy efficiency of key exchange protocols, with Kerberos being the more efficient protocol. Compared to their work, we concentrate on encryption algorithms, and more specifically on AES, with key distribution left for future work.

III. Design

AES is a block cipher encryption algorithm that uses symmetrical keys for encrypting a block of plaintext and decrypting a block of ciphertext [9]. The algorithm uses a series of rounds consisting of one or more of the following operations: byte-level substitution, permutation, arithmetical operations on a finite field and XOR-ing with a given or calculated key [10]. As a general rule, the operations are handled bytewise.

AES receives as input a plaintext of 16 bytes and the encryption key, which has a variable dimension of 16, 24 or 32 bytes. The input text is processed into the output text (ciphertext) by using the given key and applying a number of transformations. Encryption and decryption are similar, except for the fact that decryption needs an extra step —it first runs a full encryption in order to obtain the modified key needed for decrypting data.

In [11], Schneier divides symmetrical encryption algorithms in two basic categories: block ciphers and stream ciphers. A block cipher encrypts a block of plaintext producing a block of encrypted data, whilst a stream cipher can encrypt plaintexts of varying sizes. This makes block ciphers prone to security issues, if used to encrypt plaintexts longer than the block size, in a naïve way, mainly because patterns in the plaintext can appear in the ciphertext.

A more secure way to encrypt data with a block cipher can be achieved by combining the encryption algorithm with a few basic operations, in a *mode of operation*. It is worth mentioning that the operations are not directly securing data. This is the responsibility of the block cipher. Still, they should not compromise the security provided by the cipher.

A. Electronic Code Book (ECB)

The ECB mode of operation receives blocks of plaintext, respectively ciphertext, and a key and produces corresponding blocks of ciphertext, respectively plaintext. One property of this mode of operation is that two blocks of plaintext, encrypted with the same key, will result in two identical blocks of ciphertext. ECB is the most simple mode of operation. However, one major drawback is that it does not hide data patterns, meaning that identical ciphertext blocks imply the existence of identical plaintext blocks.

B. Cipher Block Chaining (CBC)

The CBC mode of operation takes as input parameters the plaintext, respectively the ciphertext, the key and an initialization vector (IV). One property of CBC is that two encrypted blocks are identical only if their respective plaintexts have been encrypted using the same key and the same IV. Unlike ECB, CBC has link dependencies, as its basic chaining mechanism makes the ciphertext blocks dependent on previously encrypted data. This, coupled with a randomly chosen IV, ensures that identical plaintext blocks will be encrypted to different ciphertext blocks.

C. Cipher Feedback (CFB)

The CFB mode of operation is very similar to CBC regarding its input parameters and the operations it performs. The main difference between them lies in the fact that CBC works as a block cipher, while CFB can be used as a stream cipher. Unlike CBC, CFB can encrypt variable-length blocks (which are not restricted to 16 bytes). The properties of this mode of operation are similar with the ones of CBC. One key difference between the two can be observed at the implementation level: CFB uses only the encryption primitive of the underlying block cipher, both for encrypting and for decrypting data.

D. Counter (CTR)

The CTR mode of operation also produces a stream cipher. The IV used in CBC and CFB is now associated with the starting value of a counter, which is incremented and used to encrypt each block. In this mode, the output from a previous block is not used for obtaining the input to the current block. In order for the described system to work, a generator is needed on both sides of the communication. The generators have to remain synchronized in order to produce the same stream of data on both sides. A disadvantage of this mode of operation is the possible desynchronization of the communicating entities. This results in the incorrect decryption of all subsequently received data.

IV. Implementation

A practical example would be a wireless sensor network, which transmits data gathered from three types of sensors: temperature, humidity and luminosity. Because of privacy and integrity concerns all data must be encrypted during transmission. The working platform for this scenario is based on the Sparrow v3.2 node [12]. Its technical specifications are:

- CPU: ATmega128RFA1, 16MHz
- Memory: 128KB flash, 16KB RAM
- Bandwidth: up to 2Mbps
- Programming: C/C++

The ATmega128RFA1 microcontroller is actually a SoC (System on Chip) which incorporates a radio transceiver compatible with the IEEE 802.15.4 standard [13]. It offers, among other things, a relatively low energy consumption (mostly in sleep states), a FIFO buffer of 128 bytes for receiving and transmitting data, a partial hardware implementation of the MAC protocol and support for AES-128.

This microcontroller facilitates secured data transmissions by incorporating a hardware acceleration module which implements the AES algorithm. The module is capable of encrypting and decrypting data in a fast track way, as most of the functions are implemented directly in hardware. It is compatible with the AES-128 standard (the key is 128 bits long) and supports encryption and decryption for ECB mode, but only encryption for CBC mode. The input to these operations consists of the plaintext/ciphertext block and the encryption key. Note that for decryption, the extra round needed by AES to compute the decryption key is performed automatically. Other modes of operation are not supported by the hardware.

As we already stated in the previous sections, energy consumption is the main issue and challenge for WSNs. In order to obtain the best approach for ensuring confidentiality with a minimum energy use, we implemented and compared AES-128, coupled with the ECB, CBC, CFB and CTR modes of operation. All four modes have both a hardware and a pure software implementation. Since only ECB has a full hardware implementation, for the other modes we used a hybrid approach, combining the hardware part from ECB with software implementations for the remaining operations. We also refer to these hybrids as hardware implementations. For the pure software implementation we used an optimized version of AES, called TableLookupAES [6].

V. Evaluation

A. Experimental setup

To measure the energy consumption of our implementation, we perform two kinds of measurements: the time required (t) and the current drawn by the node (I) while encrypting/decrypting. Using the formula $E = P \cdot t$, where $P = U \cdot I$ is the power required by the node, we can compute the energy consumed by the algorithm, be it implemented in software, in hardware or using a hybrid approach. We ensure a constant voltage U using a voltage regulator, as explained in the next subsection.

In certain applications, the latency of encrypting/decrypting a given payload might be more important than the energy consumed. For this reason, this section also presents the timing results of the different solutions, independent of the energy measurements. As we later show, the current drawn by the node using both software and hardware security approaches is practically the same. Thus, the time taken is a sufficient metric for relative comparisons between the different solutions.

1) Current measurement: For the purpose of measuring the energy consumption of the Sparrow sensor node during our experiments, we built a current sensing circuit based on the INA 193 current shunt monitor.

Fig. 1 presents the circuit we designed. Power is provided by a $3.3V$ voltage regulator, which ensures a constant voltage regardless of the current drawn by the circuit. A shunt resistor connected in series with the Sparrow node acts as a current sensor. The voltage drop on the resistor is directly proportional with the current drawn by the circuit. This has two implications. On the one hand, the chosen resistor value must be small enough not to disturb the rest of the circuit (e.g. by incurring a big voltage drop). On the other hand, the same value has to be big enough so that the expected currents register a voltage drop that can be sensed with enough precision. In order to improve the measurement precision and sensitivity, without the drawbacks of a big resistor value, we employ a INA 193 current shunt monitor, which provides a constant gain of $20V/V$ on the input voltage drop, and a 4.99Ω precision resistor with a tolerance of 0.01%. The output of the current sensing circuit is connected to a Metrix OX 5042 oscilloscope which we used to monitor the current drawn by the node during the different encryption/decryption operations. Determining the current is as simple as dividing the voltage shown on the oscilloscope by the current shunt monitor gain ($20V/V$) and the shunt resistor value (4.99Ω).

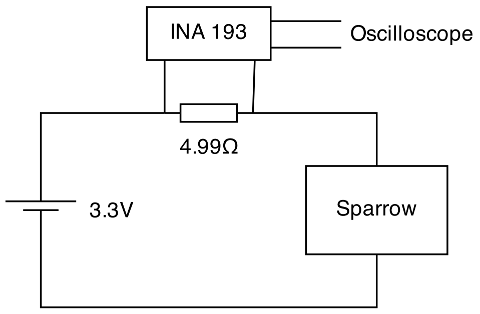


Fig. 1. Current measurement setup

2) Time measurement: Using the oscilloscope, we also measure the time required for each encryption/decryption operation. The oscilloscope has a function that accurately measures pulse duration. We create a pulse lasting for the duration of the operation by setting a GPIO pin before the start of the operation and clearing it after it ends. Using this method, we can measure the duration of an operation with minimal overhead: 1 bit set instruction and 1 bit clear instruction, each taking 2 cycles.

Although the proposed measurement scheme is precise, it has the disadvantage of requiring manual intervention. The available oscilloscope cannot be interfaced with a PC, so a measurement point is obtained by uploading a program which encrypts a hardcoded message length in a loop, reading the information from the oscilloscope and repeating the process for all message lengths.

In order to automate the time measurements, we resorted to a software implementation running along side the encryption/decryption operation, that measures the time required. To keep overhead to a minimum, our solution employs the hardware timer module available on the ATmega128RFA1 to count the number of cycles taken by the operation. Each operation is measured by sampling a counter before and after the operation and taking the difference of the two values. The count is then converted to a time value given that the microcontroller operates at $16MHz$.

This time measurement solution allowed us to automate the whole process of evaluating the algorithms for different message sizes. A small overhead can be observed between the software based time measurement and the oscilloscope based one, but the relative difference between the algorithms is unaffected. If absolute numbers are required, the software-based measurements can be corrected by noticing that the overhead increases linearly with the message size when compared with the oscilloscope measurements.

B. *Results*

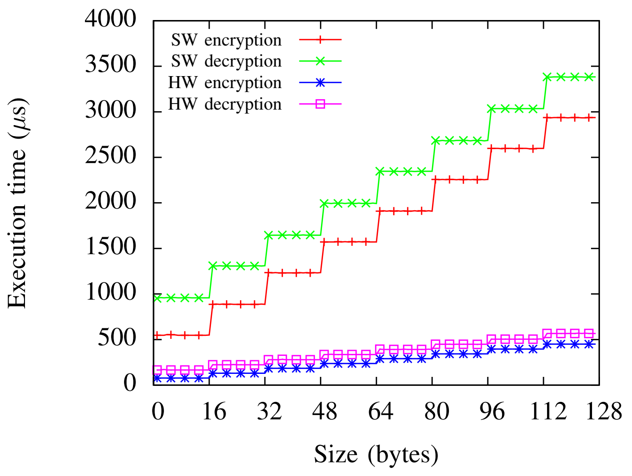
We conducted multiple experiments, to evaluate both the time taken and the energy consumed by AES encryption/decryption operations. We measured our hardware assisted implementation against the pure software implementation based on look-up tables.

1) Time experiments: We started of with measuring the difference between the optimized software implementation and our hardware assisted implementation for each of the 4 studied modes of operation. For each type of implementation and operation mode, we measured the time taken by an encryption operation and a decryption operation on varying message lengths. We used message lengths from 1 byte to 127 bytes, which is the maximum packet size allowed by the transceiver and the 802.15.4 standard.

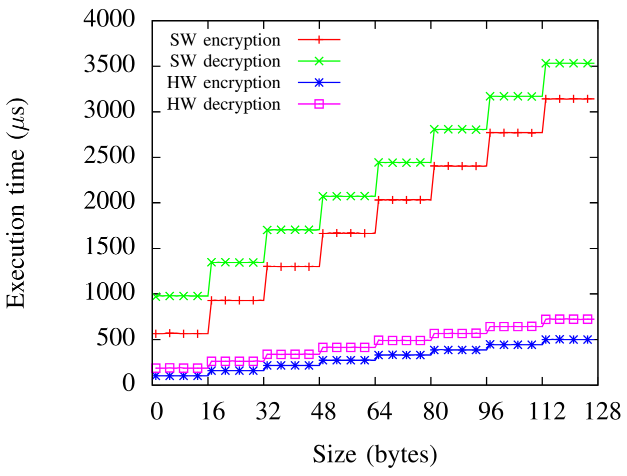
As can be seen in figure 2, the hardware assisted implementation easily outperforms the optimized software implementation. The staircase shape of the graph is easily explained by the requirement of every block cipher, including AES, to operate on multiples of the block size. Plaintext sizes that are not a multiple of the block size need to be padded, thus still incurring the cost of an entire block.

The difference in performance varies between ∼6.5x for the ECB mode, which is fully supported in hardware, down to ∼3.8x for the CFB and CTR modes, which are only partially supported in hardware through the AES single block encryption primitive. The difference in performance between the optimized software implementation and our hardware assisted implementation is summarized in table I.

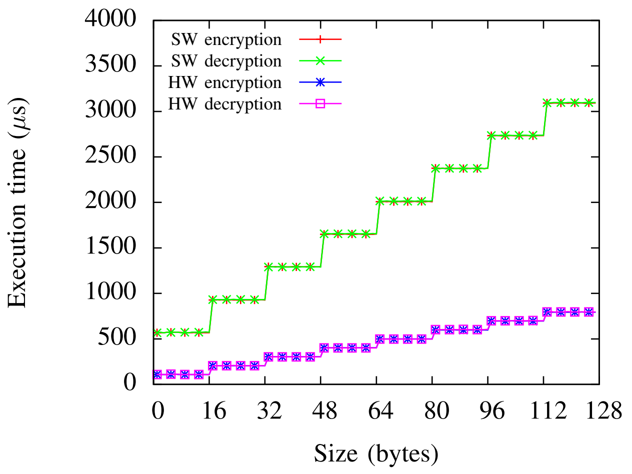
For the ECB and CBC modes we can also observe (figures 2a and 2b) the extra preparation step needed by the single block decryption primitive, which makes decryption slightly



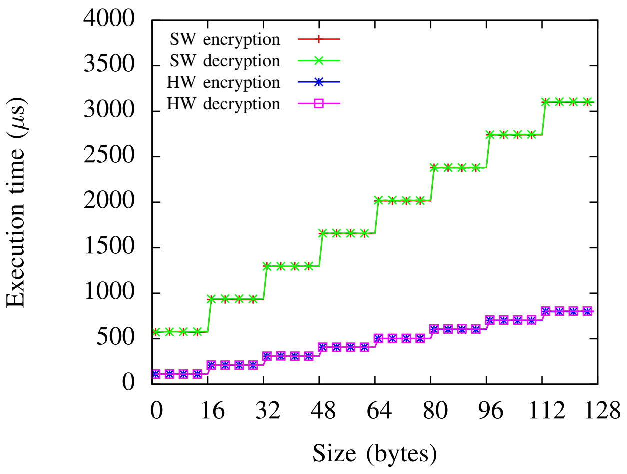
(a) ECB mode



(b) CBC mode



(c) CFB mode



(d) CTR mode

Fig. 2. Comparison between software and hardware AES implementations

	encryption	decryption
ECB	4.61x - 6.49x	5.59x - 6.02x
CBC	5.51x - 6.28x	4.86x - 5.28x
CFB	3.87x - 5.29x	3.86x - 5.34x
CTR	3.87x - 5.17x	3.85x - 5.17x

TABLE I. EXECUTION SPEED-UP HARDWARE VS. SOFTWARE

more time consuming than encryption. No difference can be observed (figures 2c and 2d) between encryption and decryption for the CFB and CTR modes, because they only use the encrypt primitive of AES for both encryption and decryption, albeit with some extra software processing.

Figure 3 compares the hardware assisted implementations of the 4 modes against each other during encryption and decryption. For encryption, ECB has the lowest runtime for all sizes, which was to be expected, as it does no extra operations on the output of the encrypt primitive to mask patterns in the plaintext. CBC is slightly worse, as it adds a XOR operation, which is implemented by the hardware accelerator, but better than CFB and CTR, which have no hardware support, except for the encryption primitive. For decryption, CFB and CTR have a slight advantage over ECB and CBC for small sizes, as they only use the encrypt primitive, which has a smaller setup time than the decrypt primitive. This advantage is lost at around 32 bytes with respect to ECB and at around 64 bytes with respect to CBC. From the plots we can also observe that the extra software processing done on top of the AES encrypt primitive by CFB and CTR is similar in overhead, for both encryption and decryption.

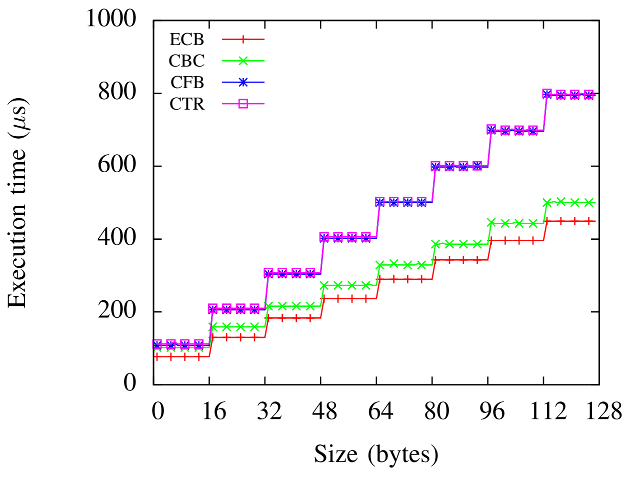
If we look at the cumulated time of both encryption and decryption, CFB and CTR still hold an advantage up to 32 bytes with respect to CBC. Thus, for small message sizes, as it usually happens in WSNs, it might be more efficient to use the CFB or CTR modes even if they are not completely accelerated in hardware.

2) Energy experiments: For energy consumption we concentrated our efforts on determining the cost of using AES in CFB mode. We chose this mode based on the fact that the timing measurements showed it to be the best encryption/decryption mode for small message sizes, similar to those that are commonly found in WSNs. We only performed measurements for message encryption, as decryption is identical in terms of the code which is ran. We measured the cost of doing the encryption in software as well as the cost of using our hardware accelerated implementation. For completeness, we also measured the cost of an empty processing loop to compare against the two encryption implementations.

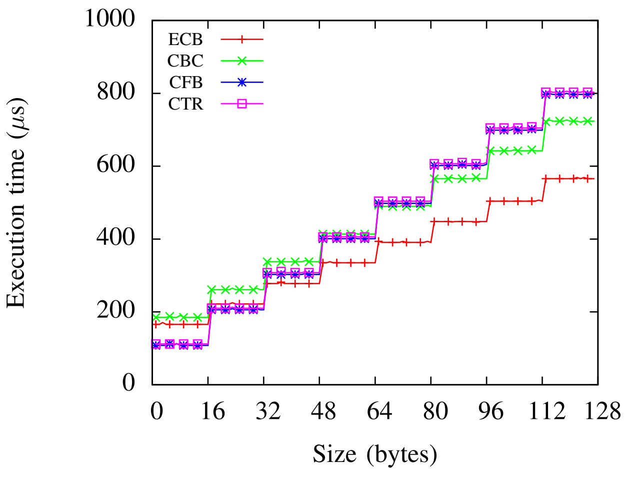
In our experiments, we used the measurement circuit described in subsection V-A1 to measure the base and peak currents during encryption, as well as the voltage and duration of the operation, as reported by the oscilloscope. As with the timing measurements, we performed the experiment for different message size, from 1 byte to 127 bytes. The oscilloscope was configured to report the mean over 16 samples in order to obtain the average energy consumption of the device. An instantaneous energy consumption is hard to obtain and is irrelevant when considering the long time operation of the node.

Using the raw current and voltage measurements, we plot the average power drawn with respect to the encryption size. As can be seen in figure 4a, the software and the hardware solutions draw equal amounts of power. Furthermore, this average power is independent of the plaintext size and is only slightly higher than the average power drawn by the empty processing loop.

If we plot the average energy consumed by the encryption operation (figure 4b), we see a linear increase in energy consumption with increasing plaintext size. Using the timing measurements performed in the previous subsection and the average power values from figure 4a, we can also derive the average energy consumption for every mode, operation and plaintext size, not just for encryption in CFB mode. This can be done by adjusting for the overhead induced by the software timer, using a correction factor deduced from correlating the oscilloscope timings with the internal timer timings.

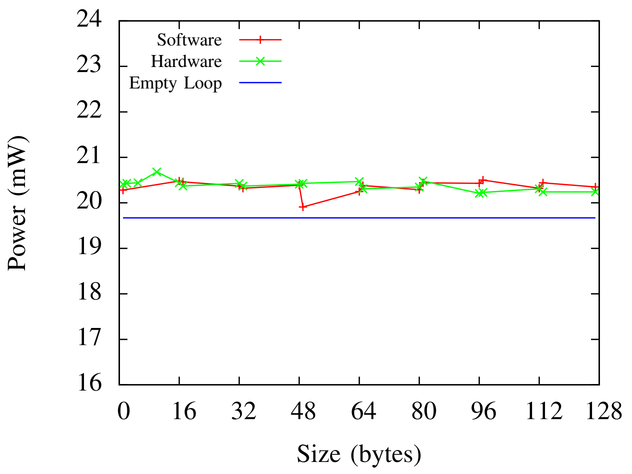


(a) Encryption

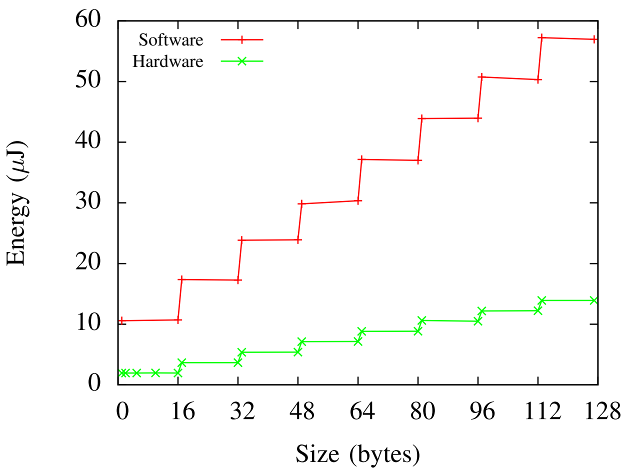


(b) Decryption

Fig. 3. Comparison between modes of operation with hardware acceleration



(a) Average power consumption



(b) Average energy consumption

Fig. 4. Power and energy consumption of AES encryption in CFB mode

VI. Conclusion

In this paper an evaluation of the cost of adding AES-128 encryption to WSN communications has been presented. Both the time penalty as well as the more important (from the point of view of a WSN) energy penalty have been analyzed, for multiple modes of operation: ECB, CBC, CFB and CTR and for two implementations: a pure software implementation, based on the optimized table lookup AES and a hardware accelerated implementation, that uses the AES hardware module of the ATmega128RFA1 microcontroller.

We showed how the AES hardware module in the ATmega128RFA1 microcontroller can be used to implement other modes of operation than the ones supported natively. Our solution uses a hybrid approach that runs some operations in hardware and emulates the missing ones in software. Using this approach, we implemented CBC decryption, as well as two full modes of operation for AES, CFB and CTR, which do not have direct hardware support.

We presented a methodology of accurately measuring the power consumption using low cost components and a way of determining the encryption/decryption duration using only the wireless node itself. We compared the different modes of operation and concluded that except for the unsecure ECB mode, CFB and CTR are better overall alternatives for the small message sizes (below 32 bytes) usually exchanged in WSNs. This is true even though the hardware accelerator has native support for the CBC mode and it relates to the way decryption works for CBC.

We also built on the work of Zhan [6] and showed that the newer ATmega128RFA1 microcontroller with an integrated transceiver, used in the Sparrow v3.2 node, can reduce both the duration and the energy consumption of AES operations. This is in contrast to work done on previous sensor nodes, that used a separated microcontroller and transceiver and which had a higher energy cost when running the encryption in hardware as opposed to using a pure software implementation.

References

[1] H. Karl and A. Willig, *Protocols and architectures for wireless sensor networks*. John Wiley & Sons, 2007. ISBN 978-0-470-09510-2

[2] D. W. Carman, P. S. Kruus, and B. J. Matt, “Constraints and approaches for distributed sensor network security (final),” DARPA project report, NAI Labs, Cryptographic Technologies Group, Trusted Information System, Tech. Rep. 1, 2000.

[3] Y. Wang, G. Attebury, and B. Ramamurthy, “A survey of security issues in wireless sensor networks,” 2006. doi: 10.1109/COMST.2006.315852

[4] J. Sen, “Routing security issues in wireless sensor networks: attacks and defenses,” in *Sustainable Wireless Sensor Networks*, W. Seah and Y. K. Tan, Eds. InTech, 2010. doi: 10.5772/663 pp. 279–309.

[5] J. Lee, K. Kapitanova, and S. H. Son, “The price of security in wireless sensor networks,” *Computer Networks*, vol. 54, no. 17, pp. 2967–2978, 2010. doi: 10.1016/j.comnet.2010.05.011

[6] F. Zhang, R. Dojen, and T. Coffey, “Comparative performance and energy consumption analysis of different aes implementations on a wireless sensor network node,” *International Journal of Sensor Networks*, vol. 10, no. 4, pp. 192–201, 2011. doi: 10.1504/IJSNET.2011.042767

[7] Y. W. Law, J. Doumen, and P. Hartel, “Survey and benchmark of block ciphers for wireless sensor networks,” *ACM Transactions on Sensor Networks (TOSN)*, vol. 2, no. 1, pp. 65–93, 2006. doi: 10.1145/1138127.1138130

[8] G. De Meulenaer, F. Gosset, O.-X. Standaert, and O. Pereira, “On the energy cost of communication and cryptography in wireless sensor networks,” in *Networking and Communications, 2008. WIMOB’08. IEEE International Conference on Wireless and Mobile Computing,*. IEEE, 2008. doi: 10.1109/WiMob.2008.16. ISBN 978-0-7695-3393-3 pp. 580–585.

[9] J. Daemen and V. Rijmen, “The block cipher rijndael,” in *Smart Card Research and Applications*. Springer, 2000. doi: 10.1007/10721064_26 pp. 277–284.

[10] W. Stallings, *Cryptography and Network Security - Principles and Practice, Fifth Edition*. Pearson Education, 2011. ISBN 978-0-13-609704-4

[11] B. Schneier, *Applied cryptography: protocols, algorithms, and source code in C*. John Wiley & Sons, 1996. ISBN 978-0471117094

[12] A. Voinescu, D. Tudose, and D. Dragomir, “A lightweight, versatile gateway platform for wireless sensor networks,” in *Networking in Education and Research, 2013 RoEduNet International Conference 12th Edition*. IEEE, 2013. doi: 10.1109/RoEduNet.2013.6714202 pp. 1–4.

[13] *8-bit AVR Microcontroller with Low Power 2.4GHz Transceiver for ZigBee and IEEE 802.15.4*, ATmega128RFA1, Atmel.